Calculation and Analysis

Models:

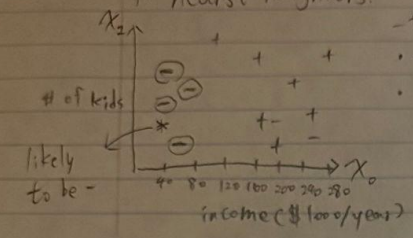
Models

Supervised learning

K-nearest neighbors: (# of neighbors ex: 4)

euclidean distance $d = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$

- find distance & classify points
- + means own a car
- − means no car



Naive Bayes:

Dataset:

Has covid?	+	−	Total
Y	531	6	537
N	20	9443	9463
Total	551	9449	

Q_2: (P) of having covid given a positive test? ($P(\text{covid} \mid +)$)

ans: 531/551

Bayes Rule:

★ $P(A \mid B) = \dfrac{P(B \mid A) \cdot P(A)}{P(B)}$

condition

ex: P(false positive) = 0.05 → $P(+ \mid \text{no disease})$

P(false negative) = 0.01 → $P(- \mid \text{disease})$

P(disease) = 0.1

	+	−
disease	0.99	0.01
no	0.05	0.95

$P(\text{disease} \mid (+)\text{test}) = ? = \dfrac{P(+ \mid \text{disease}) \cdot P(\text{disease})}{P(+)} = \dfrac{0.99 \times 0.1}{0.99 \cdot 0.1 + 0.05 \cdot 0.9} = 0.6875$ or 68.75%

$P(+) = P(+ \mid \text{disease}) \cdot P(\text{disease}) + P(+ \mid \text{no disease}) \cdot P(\text{no disease})$

posterior, feature vector, likelihood, prior

★ $P(C_k \mid x) = \dfrac{P(x \mid C_k) \cdot P(C_k)}{P(x)}$

$P(C_k)$: probability of class k in general

$P(x)$: evidence

$\prod_{i=1}^{n} P(x_i \mid C_k)$

$P(x_1, x_2 \cdots x_n \mid C_k) = P(x_1 \mid C_k) \cdot P(x_2 \mid C_k) \cdots P(x_n \mid C_k)$

in naive Bayes we assume $x_1 \sim x_n$ are all independent

C: different categories (class)

ex: C_1 = cat

C_2 = dog

⋮

C_k = hippo

Bayes Rule

$\dfrac{P(x_1, x_2 \cdots x_n \mid C_k) \cdot P(C_k)}{P(x_1, x_2 \cdots x_n)} \Rightarrow P(C_k \mid x_1, x_2 \cdots x_n) \propto P(x_1, x_2 \cdots x_n \mid C_k) \cdot P(C_k)$

$P(x_1, x_2 \cdots x_n)$: constant (no impact)

★ $P(C_k \mid x_1, x_2, \ldots x_n) \propto P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$

Probability of the class C_k given input 1~n

ex: C_k is whether we go out today given x_1 = weather, x_2 = temprature

MAP

Application of Bayes to predict y value:

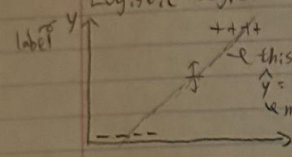
$\hat{y}$ (predicted y): $\hat{y} = (\text{argmax})\, P(C_k \mid x_1, x_2, \dots x_n) = \text{argmax}\, P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)$

take k that maximize

$k \in \{1, k\}$ means k from $1 \sim k$

- The above technique is called MAP: Maximum A Posteriori: pick a k that maximize

Logistic Regression (classify using regression):



this regression line isn't good (bc) we are classifying

$\hat{y} = mx + b$

not true

x_0 ← one of feature

$(-\infty \sim \infty)$ but p has to be $(0 \sim 1)$ => We set odds $= \frac{p}{1-p}$

true probability: $p = mx+b \Rightarrow \frac{p}{1-p} = mx+b$ but this can < 0

we then $\Rightarrow \ln\left(\frac{p}{1-p}\right) = mx+b$

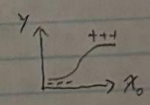
now solve for $p \Rightarrow e^{\ln\left(\frac{p}{1-p}\right)} = e^{(mx+b)}$

$$p = (1-p)\, e^{mx+b}$$

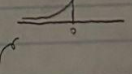
$$p = e^{mx+b} - p e^{mx+b}$$

$$p(1+e^{mx+b}) = e^{mx+b}$$

$$p = \frac{e^{mx+b}}{1+e^{mx+b}}$$



↑ which fits better btw/ $(0, 1)$



we want 1 as numerator on top $\Rightarrow$

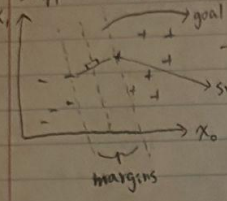
$$p = \frac{e^{mx+b}}{1+e^{mx+b}} \times \frac{e^{-(mx+b)}}{e^{-(mx+b)}}$$

$$p = \frac{1}{1+e^{-(mx+b)}} \Rightarrow \text{sigmoid function } S(y) = \frac{1}{1+e^{-y}}$$

$$\frac{1}{1+e^{-(mx+b)}} = S(mx+b)$$

- when we have only one feature x: simple logistic Regression
- when $x_0, x_1, x_2 \dots x_n \Rightarrow$ multiple logistic Regression

Support-Vector Machines (SVM):

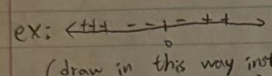
ex: 

goal is to find a line that best divide (SVM hyper plane)

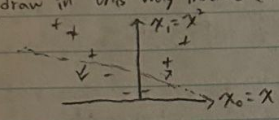
- we want to find a line that has a bigger margins

support vectors

- issue: not robust to outliers

ex: 

(draw in this way instead (kernal trick)



Back Propagation (minimize loss):

→ Gradient descent:

steepness ★ ex: $\vec{w} = \begin{bmatrix} a \\ b \end{bmatrix}$ $\nabla_{\vec{w}} f = a\frac{\partial f}{\partial x} + b\frac{\partial f}{\partial y} = \begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \vec{w} \cdot \nabla f$

definition $\frac{\partial f}{\partial x}(a,b) = \lim_{h \to 0} \frac{f(a+h,b) - f(a,b)}{h}$

ex: $\nabla_{\vec{v}} f = ?$

$\frac{\partial f}{\partial x}(\vec{a}) = \lim_{h \to 0} \frac{f(\vec{a} + h\hat{i}) - f(\vec{a})}{h}$ $\nabla_{\vec{v}} f(\vec{a}) = \lim_{h \to 0} \frac{f(\vec{a} + h\hat{v}) - f(\vec{a})}{h}$ $\hat{i} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ unit vector

- function has to be convex ⇒ $\frac{d^2 f(x)}{dx^2} > 0$ - $\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$

Gradient Descent formula: $P_{n+1} = P_n - \eta \nabla f(P_n)$

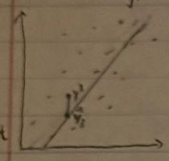
└step size (learning rate)

- use current point's gradient to calculate next point
- the smaller the learning rate ⇒ longer GD converges or may reach Max iteration before optimum point

★ saddle point is an issue

Assumption:
- linearity
- independent
- normality
- homoskedasticit

← Linear Regression:



residual $= |y_i - \hat{y}_i|$

$\sum_i |y_i - \hat{y}_i|^2$

$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n$
↳ multiple linear regression

↳ $y = b_0 + b_1 x$: simple linear regression

Evaluation or linear regression model:

① Mean absolute error $\frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$

issue ex: derivative is constant
↙ sol.

② Mean Squared error $\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$ ⇒ differniable

issue ex: $\$^2$
↙ sol.

③ Root Mean Squared $\left(\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}\right)^{\frac{1}{2}}$

back to $

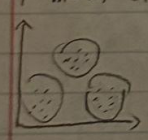
④ R^2 - coeff of determination

$R^2 = 1 - \frac{RSS}{TSS} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$

- R^2 toward 1 ⇒ good predictor

we define k first ←

K-means clustering (compute k cluster from data):



k=3

steps: ① choose 3 random points to be centroids (for 3 clusters)

② calculate distance b/w points & centroid, assing points to closet centroid ← Expectation step

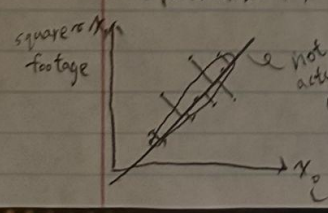
③ compute new centroids ← Maximization

stop when centroids are the same

- Expectation Maximization

Principal Component Analysis:
- dimensionality reduction

- component (direction in space) w/ largest variance



← not linear regression
actually called dimension
① minimize projection residules
② maximize variance

Softmax Activation function is continuously differentiable function. This makes it possible to calculate the derivative of the loss function with respect to every weight in the neural network. This property allows the model to adjust the weights accordingly to minimize the loss function (model output close to the true values).

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e\text{-}24$ (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

Loss function(optimize the model while training, loss decreases as performance gets better):

Crossentropy(used for classification model, take the output probabilities (P) and measure the distance from the truth values):

Binary cross-entropy is often calculated as the average cross-entropy across all data examples, that is,

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1-t_j)\log(1-p_j)\right]\right]$$

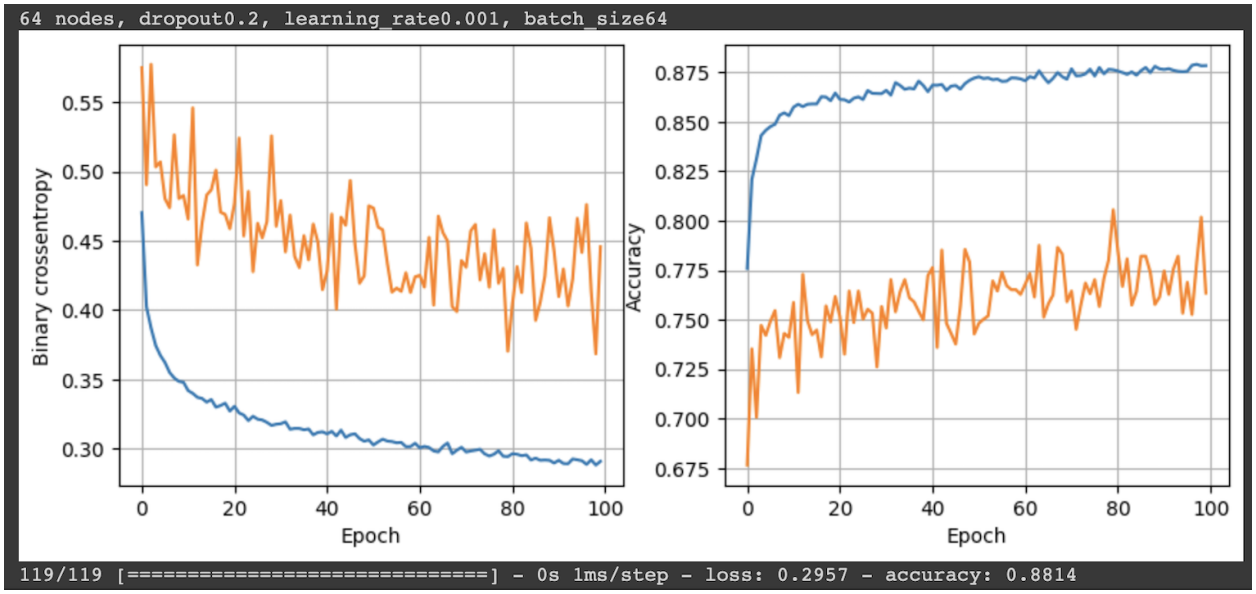
for N data points where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the ith data point.

For binary classification (a classification task with two classes — 0 and 1), we have binary cross-entropy defined as

$$\begin{aligned} L &= -\sum_{i=1}^{2} t_i \log(p_i) \\ &= -[t_1 \log(p_1) + t_2 \log(p_2)] \\ &= -[t \log(p) + (1-t)\log(1-p)] \end{aligned}$$

where t_i is the truth value taking a value 0 or 1 and p_i is the Softmax probability for the ith class. Since we have two classes 1 and 0 we can have $t_1 = 1$ and $t_2 = 0$ and since $p's$ are probabilities then $p_1 + p_2 = 1 \implies p_1 = 1 - p_2$. For the convenience of notation, we can then let $t_1 = t, t_2 = 1 - t, p_1 = p$ and $p_2 = 1 - p$.

Equation 3: Mathematical Binary Cross-Entropy.



The picture above shows that my model is working since the value of the loss function is decreasing and the accuracy is increasing.

Support Vector Classification:

Finding margins:

Equation of a Hyperplane in 2 dimensions

$$\beta_0 + \beta_1 \mathbf{X_1} + \beta_2 \mathbf{X_2} = 0$$

Equation of a Hyperplane in p-dimensions

$$\beta_0 + \beta_1 \mathbf{X_1} + \ldots + \beta_p \mathbf{X_p} = 0$$

We have points on the hyperplane:

$$\mathbf{X} = [\mathbf{X_1}, \ldots, \mathbf{X_p}]^T$$

where $\mathbf{X}$ is a feature vector

We can use this to make classifications by considering one class defined by:

$$\beta_0 + \beta_1 \mathbf{X_1} + \ldots + \beta_p \mathbf{X_p} > 0$$

With the other class defined by:

$$\beta_0 + \beta_1 \mathbf{X_1} + \ldots + \beta_p \mathbf{X_p} < 0$$

Equations defining a hyperplane and the classification rules defined by the decision boundary.

Transformation:

Kernel Trick:

In kernel methods, the data set **X** is represented by an n x n kernel matrix of pairwise similarity comparisons where the entries (i, j) are defined by the **kernel function:** k(xi, xj). This kernel function has a special mathematical property. The kernel function acts as a modified dot product. We have:

Kernel Definition

- A function that takes as its inputs vectors in the original space and returns the dot product of the vectors in the feature space is called a *kernel function*
- More formally, if we have data $\mathbf{x}, \mathbf{z} \in X$ and a map $\phi : X \rightarrow \Re^N$ then

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

is a kernel function

Equation 5-9. Kernel trick for a 2nd-degree polynomial mapping

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\, a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\, b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2 a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b} \right)^2$$

On the left-hand side, we have the dot product of the transformed feature vectors, which is equal to our 2nd-degree polynomial kernel function.

It's basically the dot product

- Solution of the dual problem gives us:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \mathbf{x}_i$$

- The decision boundary:

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0$$

- The decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0 \right]$$

- Mapping to a feature space, we have the decision:

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\phi(\mathbf{x}) \cdot \phi(\mathbf{x}_i)) + w_0 \right]$$

In the bottom equation, we replace the dot product of the transformed vectors with the kernel function.

Back Propagation(To minimize the loss):

Gradient Descent

In summary, Gradient Descent method's steps are:

1. choose a starting point (initialisation)
2. calculate gradient at this point
3. make a scaled step in the opposite direction to the gradient (objective: minimise)
4. repeat points 2 and 3 until one of the criteria is met:

- maximum number of iterations reached
- step size is smaller than the tolerance (due to scaling or a small gradient).

Propagation:

The equations above form network's forward propagation. Here is a short overview:

$$x = a^{(1)} \qquad Input\ layer$$

$$z^{(2)} = W^{(1)}x + b^{(1)} \qquad neuron\ value\ at\ Hidden_1\ layer$$

$$a^{(2)} = f(z^{(2)}) \qquad activation\ value\ at\ Hidden_1\ layer$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \qquad neuron\ value\ at\ Hidden_2\ layer$$

$$a^{(3)} = f(z^{(3)}) \qquad activation\ value\ at\ Hidden_2\ layer$$

$$s = W^{(3)}a^{(3)} \qquad Output\ layer$$

Overview of forward propagation equations colored by layer

For a single weight *(w_jk)^l,* the gradient is:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} \qquad final\ value$$

Equations for derivative of C in a single weight (w_jk)^l

(W subscript jk and superscript l)

The common part in both equations is often called *"local gradient"* and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

Equation for local gradient

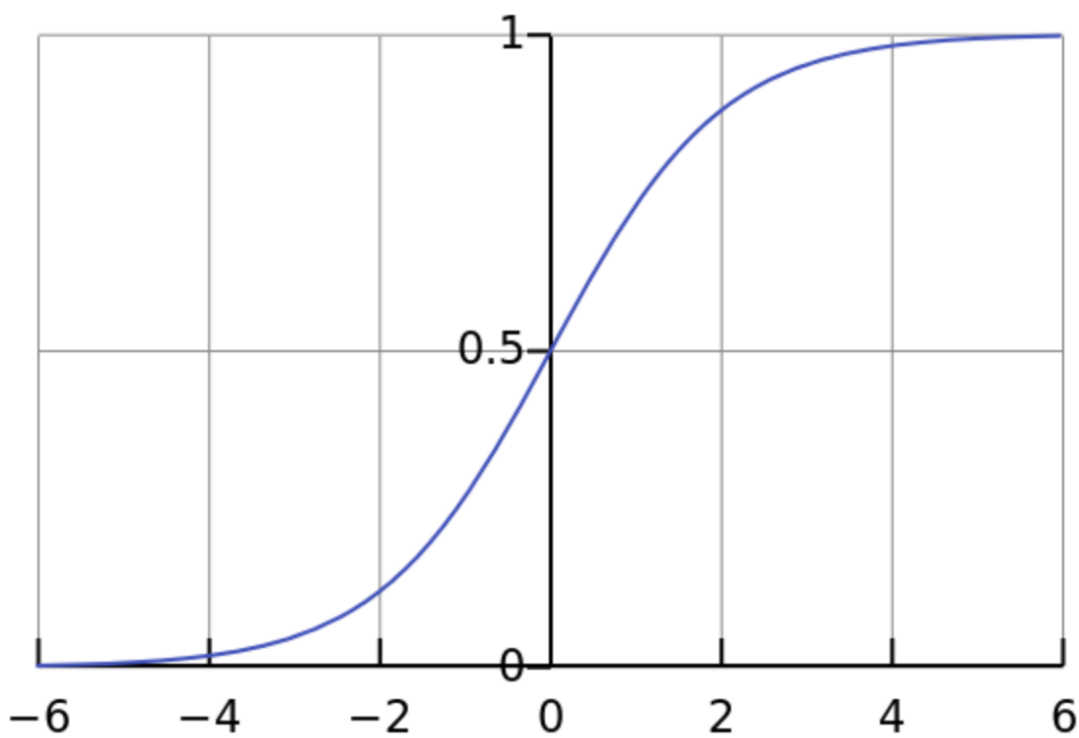
Activation Function:

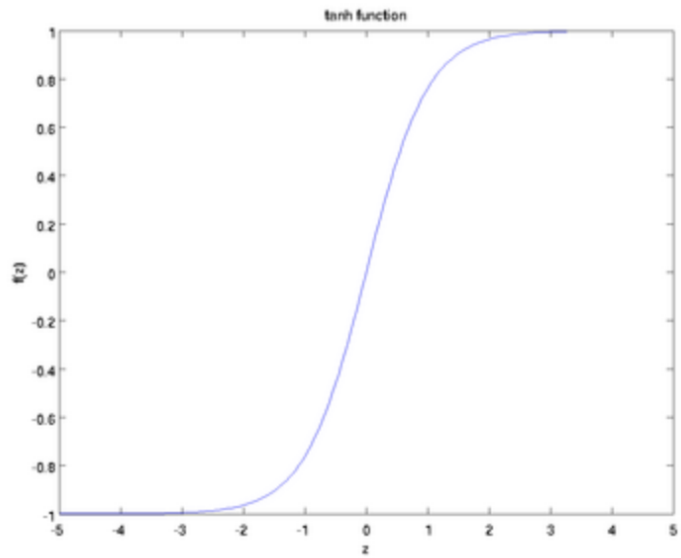
So consider a neuron.

$$Y = \sum (weight * input) + bias$$

Sigmoid Function

$$A = \frac{1}{1+e^{-x}}$$

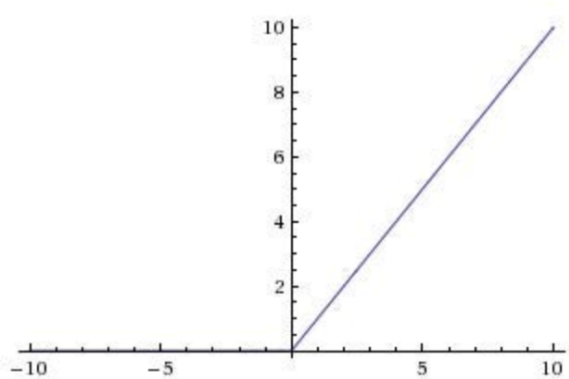




$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

$$tanh(x) = 2\ sigmoid(2x) - 1$$

The ReLu function is as shown above. It gives an output x if x is positive and 0 otherwise.



Leaky Relu is to change the slope of x<0 so the gradient won't be 0